Evaluating the Quality of UCP-Based Framework using CK Metrics

Zhamri Che Ani, Nor Laily Hashim,
Hazaruddin Harun
School of Computing
Universiti Utara Malaysia
06010 UUM Sintok, Kedah, Malaysia
{zhamri, laily, hazaruddin}@uum.edu.my

Shuib Basri, Aliza Sarlan
Department of Computer and Information Sciences
Universiti Teknologi PETRONAS
31750 Tronoh, Perak, Malaysia
{shuib_basri, aliza_sarlan}@utp.edu.my

***Abstract*—Software effort estimation is one of the most important concerns in the software industry. It has received much attention since the last 40 years to improve the accuracy of effort estimate at early stages of software development. Due to this reason, many software estimation models have been proposed such as COCOMO, ObjectMetrix, Use Case Points (UCP) and many more. However, some of the estimation methods were not designed for object-oriented technology that actively encourages reuse strategies. Therefore, due to the popularity of UCP model and the evolution of the object-oriented paradigm, a UCP-based framework and supporting program were developed to assist software developers in building good qualities of software effort estimation programs. This paper evaluates the quality of the UCP-based framework using CK Metrics. The results showed that by implementing the UCP-based framework, the quality of the UCP-based program has improved regarding the understandability, testability, maintainability, and reusability.**

***Keywords*—ucp-based framework, use case points, ck metrics**

I. Introduction

Software effort is defined as the person months required to make a software application [1]. This definition is close to [2] who define software effort as the number of staff days/weeks/months or even years associated with a project. Software effort estimation (SEE) can broadly be defined as the process of estimating the effort required to develop a software project [3]. It has been focused by many researchers over the past 40 years [4] and nowadays, it has become one of the most important concerns of the software industry [5]–[9].

There are many software estimation models have been proposed to improve the accuracy of effort estimate at early stages of software development [10]–[12]. However, some estimation methods were not designed to work well with object-oriented technology that introduces inheritance and actively encourages reuse strategies. SLIM [13], Checkpoint [14], PRICE-S [15], SEER [16], COCOMO II [10], ObjectMetrix [17], [18] and Use Case Points [19] are among the popular object-oriented estimation models that have been widely used in previous studies.

Use Case Points (UCP) is a software sizing and estimation method adopted from the standard Function Point (FP) method in solving the specific needs of object-oriented systems based on use cases [20], [21]. It was developed by Gustav Karner at Objectory Systems [19]. Previous studies have demonstrated that the accuracy of UCP estimations was quite close to the actual estimates [22]–[25]. Due to the popularity of UCP, in the last two decades, many UCP-based effort estimation techniques have been proposed [24], [26]–[35], either to give more options or to enhance the capability of UCP. Studies also showed that some parts of UCP-based models have similarity in estimating software effort [19], [24], [30], [32]. However, just a few of them are equipped with proper estimation tools. Most of them used MS Excel to calculate the estimates [36], [37].

To date, software developers have very little guidance to develop quality UCP-based software effort estimation using the object-oriented approach. Even though a tool known as U-EST was developed using Java programming language, the design framework was not accessible to the public [38]. The authors also did not claim that the tool was well-designed and able to be reused or extended by other developers. Therefore, a new framework for UCP-based software effort estimation was developed to promote the reusability of UCP. Without reusability, software applications are very hard to maintain or extend [39]–[42].

Therefore, this study aims to evaluate the quality of UCP-based framework using CK Metrics. The framework was designed using UML notations after identifying the class dependencies using Java programming language. The remainder of this paper is structured as follows. Section II and III provide some basic concept of the UCP-based framework and CK Metrics respectively. Section IV describes the experimental research design. Experimentation results and discussion are discussed in section V. Section VII includes conclusion and suggestion for future work.

II. The Proposed UCP-Based Framework

The UCP-based framework is defined as *a general reusable solution for UCP-based software effort estimation design*. This framework is not a finished design that can be transformed directly into source codes. It is a description or template for solving a UCP-based problem that can be used in many different design approaches [43]. This framework was formed based on four UCP-based models namely Use Case Points (UCP) [19], Adapted Use Case Points (AUCP) [30], Industrial use of Use Case Points (IUCP) [24] and Simplified Use Case Points (SUCP) [32]. Fig. 1 illustrates the UCP-based framework.

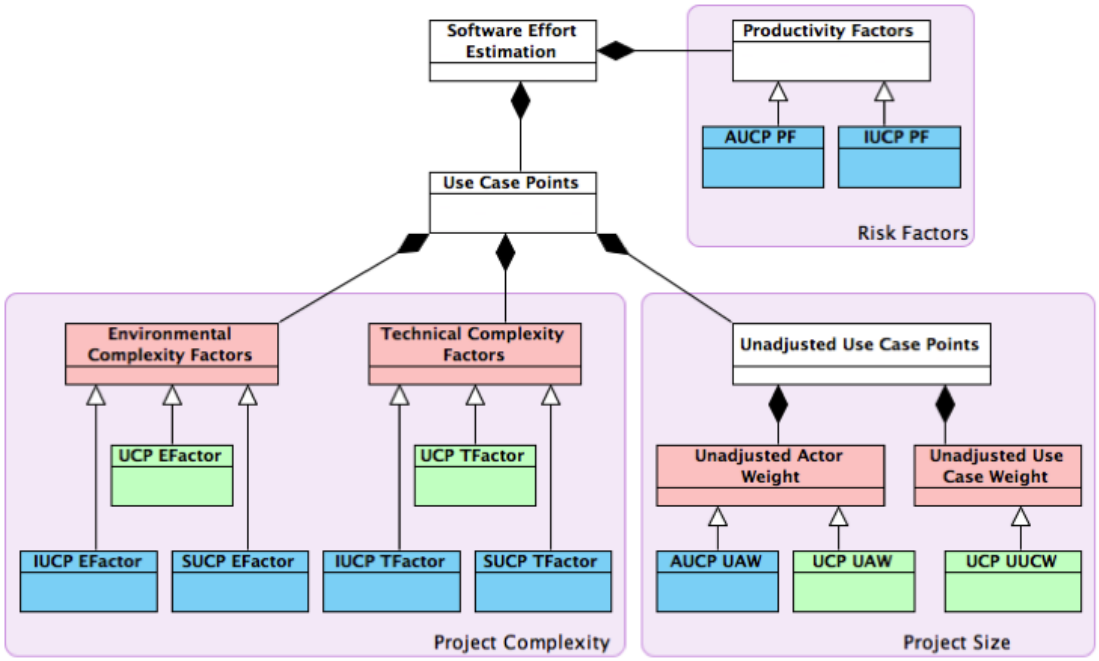

Fig. 1: A UCP-Based Framework

The UCP-based framework shows how the elements are structured, and how they work together. In other words, the UCP-based framework is more to the abstract level of software design, where the abstraction and implementation are independent. The implementations may vary dynamically [44]. This framework captured the important aspects of the UCP-based models to visualize the main packages, classes, and the relationship among them. By using this UCP-based framework, software designers can easily conduct experiments and propose the possible well-designs which can contribute to high-quality software. This framework is also useful especially in software maintenance because it suggests the high-level aspects of UCP-based requirements. By using this framework, the existing UCP-based programs can be changed or extended systematically.

The main elements of the UCP-based framework are classes and use two types of relationships namely association and generalization. The classes determine the general concept of UCP domain knowledge where every software designer familiar and understandable. Classes can be interpreted at various levels in software design. In the early stages of software design, the UCP-based framework captures more logical aspects of the problem. In the later stages, the framework can be extended to any object-oriented design decisions based on the software designer's experience and creativity. In this study, a class is drawn as a rectangle.

Overall, there are 19 classes, and 12 of them are the principal classes where the principal classes are derived from the UCP model. In general, the UCP-based framework is divided into three main components: project size, project complexity, and risk factors. These three main components are based on the estimating principle defined by Garmus and Herron [45]. Project size is composed of six classes. Five of the classes are the principal classes while `AUCP_UAW` class is the extended class. Project complexity includes four principal classes as well as four extended classes namely `IUCP_EFactor`, `SUCP_EFactor`, `IUCP_TFactor`, and `SUCP_TFactor`. Meanwhile, risk factor has only one principal class namely `Productivity_Factors` and two extended classes.

Relationships among classes are drawn as paths connecting class rectangles. Generalization shows the relationship between a more general class and a more specific class which is used for inheritance. In this framework, 11 classes are inherited from their parent classes. For instance, `AUCP_UAW` class is extended from `Unadjusted_Actor_Weight` class. Associations carry information about the relationship among objects in UCP-based domain knowledge. For instance, the `Use_Case_Points` class is associated with `Unadjusted_UseCasePoints` class. All principal classes which are captured from nine steps of UCP [36] are associated with the association relationship. This means that without these key classes the effort estimation cannot be done completely.

III. Chidamber and Kemerer (CK) Metrics

Software metrics play a major role in comparing different versions of object-oriented programs [46]. One of the most popular object-oriented metrics and the most thoroughly investigated is CK Metrics [47]. It was introduced by Chidamber and Kemerer [48] and has been a subject of discussion since the last two decades. The authors themselves and other researchers have carried out a series of experiments to improve the accuracy of the metrics. Even though it was proposed quite a long time ago, the usefulness in analyzing open-source software is still significant [49]–[52], including examining the reusability of software projects [53].

Chidamber and Kemerer proposed six metrics for evaluating the quality of software design namely Weighted Method per Class (WMC), Response For a Class (RFC), Lack of Cohesion in Methods (LCOM), Coupling Between Objects (CBO), Depth of Inheritance Tree (DIT) and Number of Children (NOC). The definition of each metric can be found in [48]. The metrics have been theoretically validated and widely accepted standard to be used as early quality indicators [52], [54]–[58]. By using these metrics, it will help software designers to make a better decision in designing any object-oriented software applications.

IV. Experimental Setting

The object-oriented approach is very important among all software practitioners. Generally, encapsulation, inheritance, and polymorphism are three main object-oriented principles that are applied throughout the whole object-oriented software engineering process [59]. Instead of these three principles, abstraction is also considered as a key element in designing software applications. By using abstraction, the complexity of a large problem can be minimized. The idea of abstraction is to identify common features in two or more classes and abstract those features out into a higher-level class [60]. A good design, each method in a well-designed class should support abstraction and encapsulation [61]. In this case, all common functionalities of UCP-based models were grouped to form reusable abstract classes.

In this experiment, a new program known as Like-UCP was developed to simulate the UCP-based framework. In order to achieve the consistency between the UCP-based framework and Like-UCP program, the Like-UCP was designed based on four object-oriented programming principles; abstraction, inheritance, encapsulation, and polymorphism. The design must ensure that, for every abstract class of Like-UCP, the subclass must implement the method from the abstract class unless the subclass is also an abstract class.

After completing the development, CK Metrics was used to measure the quality of Like-UCP program. It is impossible to obtain the Like-UCP metrics without using supporting tools. Thus, in this study, CK Java Metrics (CKJM) extended version 2.2 [62] was used to obtain the metrics. CKJM is an open-source program which was also written in Java programming language used to obtain six CK Metrics of the object-oriented programs. In many cases, no single design style can meet all quality attributes simultaneously. Software architects or designers often need to balance among quality attributes. Most of the time, a new program needs refinement, extension, generalization, or improvement [63]. Therefore, if the quality of Like-UCP program did not achieve the desired quality attributes, repetition process from designing the UCP-based framework must be done.

Then, each of the classes was analyzed, and each metric was calculated to obtain the mean value. The mean value of each metric was used as an indicator of quality measurements. Finally, the obtained results were concluded based on the relationship between CK Metrics, Object-oriented Design (OOD) measures, and quality factors. Table I and Table II show the relationship between CK Metrics and OOD measures, and the relationship between CK Metrics and quality factors respectively [64].

TABLE I: Relationship between CK Metrics and OOD Measures

OOD Measures	CK Metrics
Class	WMC, RFC, LCOM
Attribute	LCOM
Method	WMC, RFC, LCOM
Inheritance	DIT, NOC
Cohesion	LCOM
Coupling	CBO, RFC

TABLE II: Relationship between CK Metrics and Quality Factors

Quality Factors	CK Metrics
Understandability	RFC, CBO, DIT
Reusability	WMC, CBO, DIT, NOC
Testability	RFC, CBO, NOC
Maintainability	WMC, CBO
Development Effort	WMC, LCOM

V. Experimentation Results and Discussion

The Like-UCP program was developed using Java programming language to evaluate the quality of the UCP-based framework. To visualize the dependencies between classes, class diagram of Like-UCP was generated in Eclipse environment. Fig. 2 shows the class dependencies of Like-UCP program. Basically, each of the classes is a replication of UCP-based framework using the object-oriented approach. As can be seen in Fig. 2, some of the classes such as `UCP_UUCW`, `Unadjusted_UseCasePoints`, and `UseCasePoints` have high dependencies compared to other classes. In other words, these three classes are required by all the identified UCP-based models to form the UCP-based framework.

To obtain the empirical evidence of quality attributes achieved by Like-UCP program, 22 classes (19 classes derived from the UCP-based framework with three additional driver classes) were analyzed using CKJM-extended-2.2. Table III and Table IV present the descriptive statistics of Like-UCP program and the summary of all the metrics respectively. In this experiment, the total number line of codes (LOC) of Like-UCP is 2556 with the mean value 116.18. The maximum LOC is 380 used by `IUCP` class, and the minimum is 11 used by `UseCasePoints` class. The next sub-sections will discuss the obtained results and compare with the suggestions in [48].

A. Weighted Method per Class (WMC)

The number of methods and the complexity of the methods involved indicate how much time and effort is required to develop and maintain the class. The larger the number of methods in a class, the higher the potential impact on subclasses, since subclasses will inherit all the methods defined in the superclass. It was recommended that most classes should have a small number of methods, maximum up to 10 methods in a class [48]. If WMC value is one, it was recommended that to merge the class in some other classes within the same package without influencing the LCOM value, that is without affecting the abstraction and encapsulation of the classes. If WMC value is zero, the possibility of a redesign is high. If WMC over than 20, the class should be refactored to reduce the complexity of the software project. 20 WMC is consistent with the threshold value suggested by Shatnawi [65].

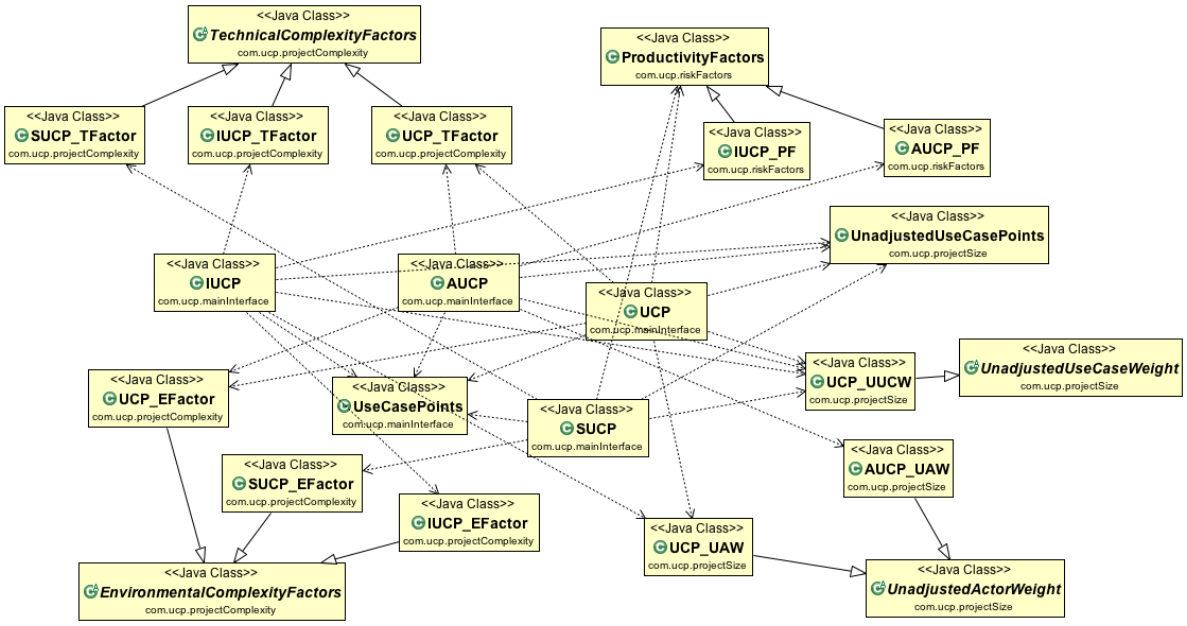

Fig. 2: Class Dependencies of Like-UCP Program

TABLE III: CK Metrics for Like-UCP Program

CLASS	LOC	WMC	RFC	LCOM	CBO	DIT	NOC
UCP	281	9	34	24	8	1	0
UseCasePoints	11	2	3	1	4	1	0
UnadjustedUseCasePoints	12	3	4	3	4	1	0
UnadjustedActorWeight	70	9	10	8	2	1	2
AUCP_UAW	24	2	5	1	2	2	0
UCP_UAW	40	2	7	1	3	2	0
UnadjustedUseCaseWeight	91	10	11	3	1	1	1
UCP_UUCW	65	3	9	1	5	2	0
EnvironmentalComplexity	171	15	16	45	3	1	3
IUCP_EFactor	86	2	13	1	2	2	0
UCP_EFactor	102	4	14	6	3	2	0
SUCP_EFactor	38	2	7	1	2	2	0
TechnicalComplexity	265	20	21	96	3	1	3
IUCP_TFactor	126	2	19	1	3	2	0
UCP_TFactor	142	4	19	6	4	2	0
SUCP_TFactor	54	2	9	1	2	2	0
ProductivityFactors	30	7	8	15	5	1	2
AUCP_PF	23	3	4	3	2	2	0
IUCP_PF	127	12	13	28	2	2	0
IUCP	380	12	46	54	9	1	0
AUCP	218	10	33	33	8	1	0
SUCP	200	8	30	18	7	1	0

TABLE IV: Summary of CK Metrics for Like-UCP Program

Metrics	Total	Mean	Maximum	Minimum
LOC	2556	116.18	380	11
WMC	143	6.50	20	2
RFC	335	15.23	46	3
LCOM	350	15.91	96	1
CBO	84	3.82	9	1
DIT	33	1.5	2	1
NOC	11	0.50	3	0

Based on the WMC statistics shown in Table IV, it can be seen that the mean value of the WMC metric is 6.50 which is less than 10. Only four classes, `TechnicalComplexity`, `EnvironmentalComplexity`, `IUCP_PF`, and `IUCP` have more than 10 but still below 20 WMC. None of the classes has zero or one WMC. These results indicate that all the classes were properly designed.

B. Response For a Class (RFC)

RFC is the number of methods that can be invoked in response to a message in a class. If RFC for a class is large, it means that there is high complexity [48]. If RFC increases, the effort required for testing will increase because the test sequence grows. The overall design complexity of the class also increases and it is difficult to maintain the classes later on. The RFC for a class should usually not exceed 50

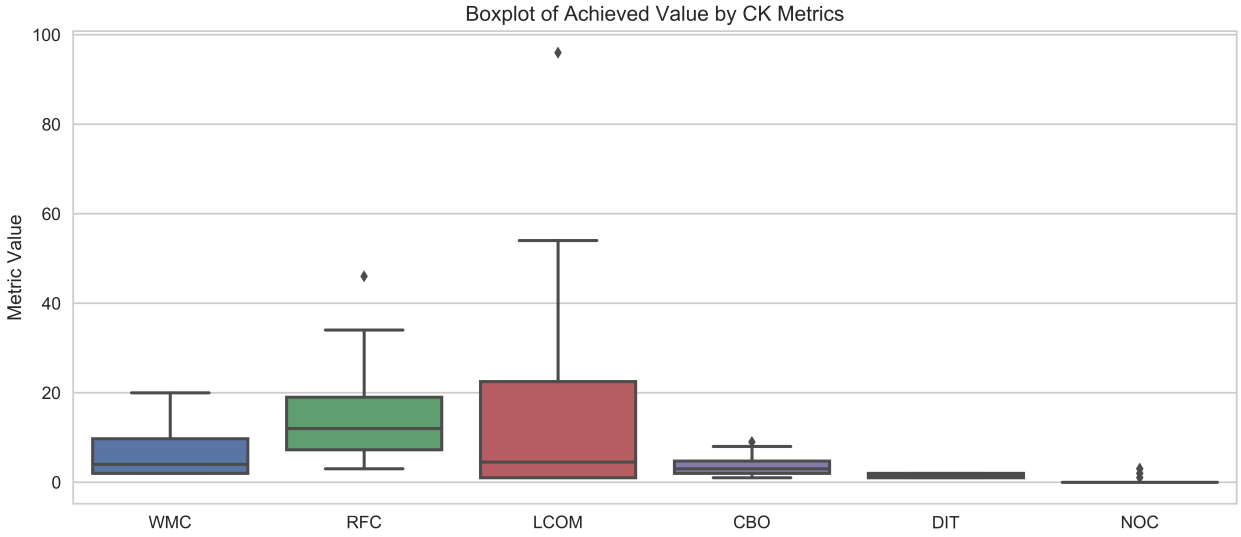

Fig. 3: Boxplot of Achieved Value by CK Metrics

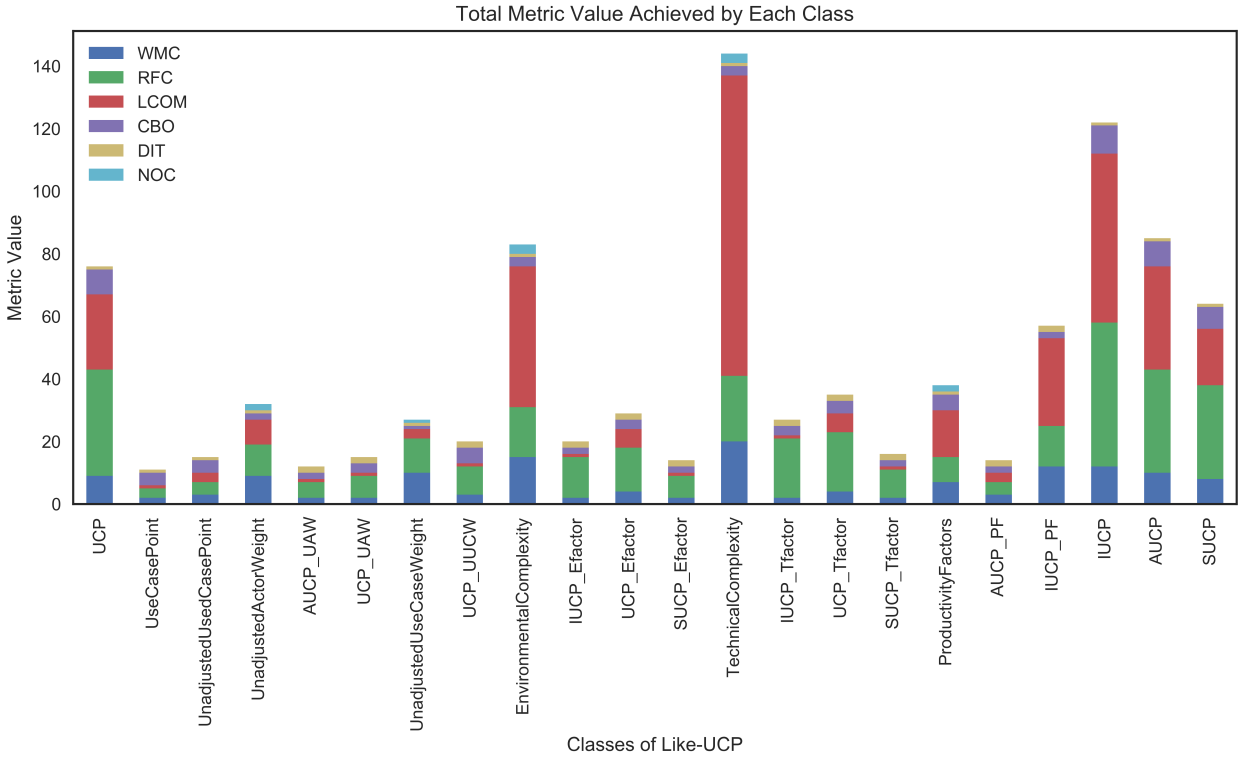

Fig. 4: Total Metric Value Achieved by Each Class

although it is acceptable to have RFC up to 100 [66]. However, Shatnawi [65] suggested that the threshold value for RFC is 40. Based on the RFC statistics shown in Table IV, it can be seen that the mean value of the RFC metric is 15.23. None of the classes exceed 50. Fig. 3 shows that only one class which is `IUCP` has more than 40 RFC. Overall, the low values of RFC indicate that the Like-UCP design is less complex.

C. Lack of Cohesion in Methods (LCOM)

The cohesiveness of methods inside a class is desirable since it promotes encapsulation and decreases the complexity of the objects. High cohesion decreases complexity, thereby decreasing the probability of errors during the development process [48]. Accordingly, a high LCOM value is a major issue in all the software projects. High LCOM value shows

poor encapsulation and abstraction at the class level. The general threshold for LCOM is classified as Good: 0, Regular: range between $1 - 20$ and Bad: > 20 [67]. Based on the LCOM statistics shown in Table IV, the mean value of LCOM is 15.91. This value indicates that most of the classes are normal and less complex. Only six classes, `UCP`, `AUCP`, `IUCP`, `IUCP_PF`, `TechnicalComplexity`, and `EnvironmentalComplexity` have greater than 20. However, as can be seen in Fig. 3, only one class which is `TechnicalComplexity` has a very high value of LCOM. Fig. 4 also shows that the total value of `TechnicalComplexity` class is also the highest among all classes of Like-UCP. The results indicate that the `TechnicalComplexity` class should be re-designed to improve the quality the overall quality of Like-UCP program.

D. Coupling Between Objects (CBO)

In general, objects should be loosely coupled, which implies there ought to be a little dependency between objects. High coupling implies that the sensitivity to changes in other classes is also high which may indicate poor class design [48]. A class which is tightly coupled will cause a large ripple effect when an object is changed and increasing the risk of regressions. Furthermore, testing all the changes will be tedious and hard to check. In contrast, the low CBO values demonstrate that most of the classes refer to a few other classes. In other words, the more independent a class is, the easier to reuse it in other applications. Therefore, it increases the understandability, efficiency, and reusability of the class design [48]. A measure of coupling is useful to determine how complex the testing of various parts of a class design are likely to be. It was suggested that classes with a CBO more than 19 should be examined to reduce coupling [65]. Based on the CBO statistics shown in Table IV, the mean value of CBO is 3.82. None of the CBO values greater than 19. The maximum value of Like-UCP is nine which is equal to the threshold value suggested by Shatnawi [65]. Therefore, it can be concluded that the possibilities of understandability, efficiency, and reusability of the class design are very high.

E. Depth of Inheritance Tree (DIT)

DIT is the length of the longest path from a subclass to the superclass in the inheritance hierarchy. A high value of DIT implies more reusability, but the complexity of the class design may increase due to more methods and classes involved [48]. Classes with bunches of subclasses need to be very carefully modified to avoid regressions in those subclasses. One conceivable solution is that the classes ought to be more abstract with a reasonable number. It was proposed that the maximum value of DIT should be less than 10 [48]. If DIT value is zero or one, it demonstrates poor reusability of the class. Based on the DIT statistics shown in Table IV, the mean value of DIT is 1.5. This value indicates that inheritance was rarely used in Like-UCP program, mainly because of the domain model. Consequently, the program did not have a deep inheritance tree. However, the maximum value of DIT is two which is evaluated as *Good/Common* by DIT threshold [68]. Therefore, with 1.5 DIT, Like-UCP program can be concluded as potentially to be reused and easy to be understood by software developers.

F. Number of Children (NOC)

Inheritance is a form of reuse. Thus, the greater the number of subclasses for a class, the greater reuse. However, a large number of subclasses may introduce inappropriate abstraction of the superclass. It is better to have depth than breadth in the inheritance hierarchy [48]. In other words, high DIT and low NOC is the perfect combination for software design. Low values of DIT and NOC are firmly indicated that reuse through inheritance may not be completely adopted [48]. Based on the NOC statistics shown in Table IV, the majority of the classes (77%) have no subclasses, and the mean value of NOC is 0.50 which is less than the mean value of DIT. Fig. 3 shows that Like-UCP program has a minimal number of outliers. The results indicate that the Like-UCP program may not be using inheritance of methods as a basis for designing classes. However, these results are consistent with the previous case studies [48]. Due to the mean value of NOC smaller than the mean value of DIT, most probably the Like-UCP classes are potential to be reused by other UCP-based programs.

VI. Threats to Validity

Each of metric tools might interpret and produce a different set of results. To avoid this problem, the replication of this study must follow the interpretation that we describe in this paper. However, we encourage readers to use different metric tools with different programming languages.

VII. Conclusion and Future Work

This study presents an evaluation of UCP-based framework using CK Metrics. In order to achieve this objective, Like-UCP program was developed using Java programming language. The program was a replication of the UCP-based framework using the object-oriented approach. Then the developed program was examined using CK Metrics to evaluate whether the Like-UCP has accomplished the desired quality standard. As can be seen in Fig. 4, `TechnicalComplexity` and `EnvironmentalComplexity` have higher metric values. Therefore, these two classes need more treatments in order to improve the quality of Like-UCP program. As mentioned earlier, `IUCP`, `AUCP`, and `SUCP` are the class drivers of Like-UCP. These classes were merely used for program simulation. Thus, the high metric values of these three classes will not affect the overall quality of UCP-based framework.

Overall, based on WMC, RFC, CBO, DIT and NOC obtained by CKJM-extended-2.2, it can be concluded that Like-UCP program was well designed. Low value of LCOM indicates that most of these classes were also less complex. Although this experiment was based on a small sample of programs, the results have provided additional evidence that by implementing the object-oriented approach, the quality of Like-UCP has improved regarding the understandability, testability, maintainability, and reusability of the program. The results also indicate that less effort was required to develop the Like-UCP program. As mentioned earlier, Like-UCP is a replication of UCP-based framework. Therefore, the achievement of these quality attributes also reflects the UCP-based framework. In other words, the UCP-based framework can also be said good qualities regarding the understandability, testability, maintainability, and reusability.

The results also show that the program did not have a deep inheritance tree due to the fewer functionalities in the domain model. However, this can be improved by implementing more UCP-based models in the program. Hence, we intend to additionally investigate this topic by utilizing other object-oriented design methodologies such as implementing SOLID Design Principles and make a comprehensive comparison against these outcomes.

ACKNOWLEDGMENT

The researchers acknowledge the financial support (Research Generation University Grant) received from University Utara Malaysia (S/O Code: 13847).

REFERENCES

[1] S. Tariq, M. Usman, R. Wong, Y. Zhuang, and S. Fong, "On learning software effort estimation," in *3rd International Symposium on Computational and Business Intelligence (ISCBI)*. IEEE, 2015, pp. 79–84.

[2] L. Laird and M. Brennan, *Software measurement and estimation: a practical approach*. New Jersey: John Wiley and Sons, 2006.

[3] L. L. Minku and X. Yao, "Which models of the past are relevant to the present? a software effort estimation approach to exploiting useful past models," *Automated Software Engineering*, vol. 24, no. 3, pp. 499–542, 2017.

[4] C. Jones, "Software cost estimation in 2002," *The Journal of Defense Software Engineering*, vol. 15, no. 6, pp. 4–8, 2002.

[5] S. Basha and D. Ponnurangam, "Analysis of empirical software effort estimation models," *International Journal of Computer Science And Information Security*, vol. 7, no. 3, 2010.

[6] F. Brooks Jr, "Three great challenges for half-century-old computer science," *Journal of the ACM*, vol. 50, no. 1, pp. 25–26, 2003.

[7] M. Jørgensen and D. Sjøberg, "The impact of customer expectation on software development effort estimates," *International Journal of Project Management*, vol. 22, no. 4, pp. 317–325, 2004.

[8] S. W. I. Kuan, "Factors on software effort estimation," *International Journal of Software Engineering & Applications*, vol. 8, no. 1, pp. 23–32, 2017.

[9] A. Trendowicz, J. Münch, and R. Jeffery, "State of the practice in software effort estimation: a survey and literature review," *Software Engineering Techniques*, vol. 4980, pp. 232–245, 2011.

[10] B. Boehm, "Software engineering economics," *IEEE Transactions on Software Engineering*, vol. 10, pp. 4–21, 1984.

[11] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches-a survey," *Annals of Software Engineering*, vol. 10, no. 1-4, pp. 177–205, 2000.

[12] A. Trendowicz and R. Jeffery, *Software Project Effort Estimation: Foundations and Best Practice Guidelines for Success*. Switzerland: Springer International Publishing, 2014.

[13] L. H. Putnam and W. Myers, *Measures for excellence: reliable software on time, within budget*. Prentice Hall Professional Technical Reference, 1991.

[14] J. Capers, *Applied software measurement*. McGraw-Hill, 1996.

[15] R. Park, "The central equations of the price software cost model," in *4th COCOMO Users Group Meeting*, 1988.

[16] R. Jensen, "An improved macrolevel software development resource estimation model," in *5th ISPA Conference*, 1983, pp. 88–92.

[17] G. Adens and R. Armstrong, "Objectmetrix process," UK:TASSC, Tech. Rep., 2008.

[18] G. Adens, "Interpreting and calibrating metrics," UK:TASSC, Tech. Rep., 2009.

[19] G. Karner, "Resource estimation for objectory projects," *Objective Systems SF AB*, vol. 17, 1993.

[20] R. Clemmons, "Project estimation with use case points," *The Journal of Defense Software Engineering*, pp. 18–22, 2006.

[21] G. Banerjee, "Use case estimation framework," in *Annual IPML Conference*. Citeseer, 2004, pp. 1–12.

[22] B. Anda, H. Dreiem, D. Sjøberg, and M. Jørgensen, "Estimating software development effort based on use cases - experiences from industry," *The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, vol. 2185, pp. 487–502, 2001.

[23] B. Anda, "Comparing effort estimates based on use case points with expert estimates," *Empirical Assessment in Software Engineering (EASE 2002), Keele, UK*, 2002.

[24] E. Carroll, "Estimating software based on use case points," in *20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, San Diego, CA, USA, 2005, pp. 257–265.

[25] S. Nageswaran, "Test effort estimation using use case points," in *Quality Week 2001*, San Francisco, California, USA, 2001.

[26] M. R. Braz and S. R. Vergilio, "Software effort estimation based on use cases," in *30th Annual International Computer Software and Applications Conference, COMPSAC'06.*, vol. 1. IEEE, 2006, pp. 221–228.

[27] S. Diev, "Use cases modeling and software estimation: applying use case points," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 6, pp. 1–4, 2006.

[28] W. Fan, Y. Xiaohu, Z. Xiaochun, and C. Lu, "Extended use case points method for software cost estimation," in *International Conference on Computational Intelligence and Software Engineering.* IEEE, 2009, pp. 1–5.

[29] M. M. Kirmani and A. Wahid, "Revised use case point (re-ucp) model for software effort estimation," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 3, pp. 65–71, 2015.

[30] P. Mohagheghi, B. Anda, and R. Conradi, "Effort estimation of use cases for incremental large-scale software development," in *27th International Conference on Software Engineering*. IEEE, 2005, pp. 303–311.

[31] N. Nunes, L. Constantine, and R. Kazman, "iucp: Estimating interactive-software project size with enhanced use-case points," *IEEE software*, vol. 28, no. 4, pp. 64–73, 2011.

[32] M. Ochodek, J. Nawrocki, and K. Kwarciak, "Simplifying effort estimation based on use case points," *Information and Software Technology*, vol. 53, no. 3, pp. 200–213, 2011.

[33] K. Periyasamy and A. Ghode, "Cost estimation using extended use case point (e-ucp) model," in *International Conference on Computational Intelligence and Software Engineering.* IEEE, 2009, pp. 1–5.

[34] G. Robiolo, C. Badano, and R. Orosco, "Transactions and paths: two use case based metrics which improve the early effort estimation," in *3rd International Symposium on Empirical Software Engineering and Measurement, ESEM.* IEEE, 2009, pp. 422–425.

[35] A. Srivastava, S. Singh, and S. Q. Abbas, "Advancement of ucp with end user development factor: Aucp," *International Journal of Software Engineering & Applications (IJSEA)*, vol. 6, no. 2, pp. 1–10, 2015.

[36] Ç. Gencel, L. Buglione, O. Demirors, and P. Efe, "A case study on the evaluation of cosmic-ffp and use case points," in *3rd Software Measurement European Forum*, Rome, Italy, 2006.

[37] Z. Mansor, S. Yahya, and N. H. H. Arshad, "Review on traditional and agile cost estimation success factor in software development project," *International Journal of New Computer Architectures and their Applications (IJNCAA)*, vol. 1, no. 4, pp. 942–952, 2011.

[38] S. Kusumoto, F. Matukawa, K. Inoue, S. Hanabusa, and Y. Maegawa, "Estimating effort by use case points: Method, tool and case study," in *10th International Symposium on Software Metrics*, Washington, USA, 2004, pp. 292–299.

[39] C. Alexander, S. Ishikawa, and M. Silverstein, *A pattern language: towns, buildings, construction*. Oxford University Press, USA, 1977, vol. 2.

[40] M. Grand, *Java enterprise design patterns*. Wiley, 2002.

[41] P. Kuchana, *Software architecture design patterns in Java*. CRC Press, 2004.

[42] C. Lasater, *Design patterns*. Jones & Bartlett Learning, 2006.

[43] Z. C. Ani, S. Basri, and A. Sarlan, "A framework for designing ucp-based effort estimation," *Advanced Science Letters*, vol. 24, no. 2, pp. 995–998, 2018.

[44] K. Lano, J. L. Fiadeiro, and L. Andrade, *Software design using Java 2*. New York: Springer, 2002.

[45] D. Garmus and D. Herron, "Estimating software earlier and more accurately," *Journal of Defense Software Engineering*, vol. 15, no. 6, pp. 18–21, 2002.

[46] A. Kaur, S. Singh, D. K. Kahlon, and P. S. Sandhu, "Empirical analysis of ck & mood metric suit," *Int. Journal of Innovation, Management and Technology*, vol. 1, no. 5, pp. 447–452, 2010.

[47] M. Genero, M. Piattini, and C. Calero, "A survey of metrics for uml class diagrams," *Journal of Object Technology*, vol. 4, no. 9, pp. 59–92, 2005.

[48] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476–493, 1994.

[49] K. Johari and A. Kaur, "Validation of object oriented metrics using open source software system: an empirical study," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 1, pp. 1–4, 2012.

[50] T. Honglei, S. Wei, and Z. Yanan, "The research on software metrics and software complexity metrics," in *International Forum on Computer Science-Technology and Applications, IFCSTA'09.*, vol. 1. IEEE, 2009, pp. 131–136.

[51] S. Srivastava and R. Kumar, "Indirect method to measure software quality using ck-oo suite," in *International Conference on Intelligent Systems and Signal Processing (ISSP)*. IEEE, 2013, pp. 47–51.

[52] R. Subramanyam and M. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[53] D. P. Darcy and C. F. Kemerer, "Oo metrics in practice," *Software, IEEE*, vol. 22, no. 6, pp. 17–19, 2005.

[54] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[55] L. Cheikhi, R. E. Al-Qutaish, A. Idri, and A. Sellami, "Chidamber and kemerer object-oriented measures: Analysis of their design from the metrology perspective," *International Journal of Software Engineering & Its Applications*, vol. 8, no. 2, 2014.

[56] N. I. Churcher, M. J. Shepperd, S. Chidamber, and C. Kemerer, "Comments on" a metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 21, no. 3, pp. 263–265, 1995.

[57] M. Hitz and B. Montazeri, "Chidamber and kemerer's metrics suite: a measurement theory perspective," *Transactions on Software Engineering*, vol. 22, no. 4, pp. 267–271, 1996.

[58] G. Succi, W. Pedrycz, S. Djokic, P. Zuliani, and B. Russo, "An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite," *Empirical Software Engineering*, vol. 10, no. 1, pp. 81–104, 2005.

[59] K. Qian, X. Fu, L. Tao, and C.-w. Xu, *Software architecture and design illuminated*. Boston: Jones & Bartlett Learning, 2010.

[60] J. F. Dooley, *Software Development, Design and Coding: With Patterns, Debugging, Unit Testing, and Refactoring (2nd ed.)*. [e-book]. Retrieved from https://www.apress.com.: Apress, 2017.

[61] A. Dingle, *Software Essentials: Design and Construction*. Boca Raton: Chapman and Hall/CRC, 2014.

[62] M. Jureczko and D. Spinellis, *Using Object-Oriented Design Metrics to Predict Software Defects*, ser. Monographs of System Dependability. Wroclaw, Poland: Oficyna Wydawnicza Politechniki Wroclawskiej, 2010, vol. Models and Methodology of System Dependability, pp. 69–81.

[63] S. Lammers, *Programmers at work*. Redmond, WA: Harper & Row Publishers, Inc., 1986.

[64] U. Kulkarni, Y. Kalshetty, and V. Arde, "Validation of ck metrics for object oriented design measurement," in *3rd International Conference on Emerging Trends in Engineering and Technology (ICETET)*. IEEE, 2010, pp. 646–651.

[65] R. Shatnawi, "A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems," *IEEE Transactions on software engineering*, vol. 36, no. 2, pp. 216–225, 2010.

[66] M. Sarker, "An overview of object oriented design metrics," Master's thesis, Department of Computer Science, Umeå University, Sweden, 2005.

[67] K. A. Ferreira, M. A. Bigonha, R. S. Bigonha, L. F. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *Journal of Systems and Software*, vol. 85, no. 2, pp. 244–257, 2012.

[68] T. G. Filó, M. Bigonha, and K. Ferreira, "A catalogue of thresholds for object-oriented software metrics," in *The First International Conferences on Advanced and Thrends in Software Engineering*, 2015, pp. 48–55.